1. The image shown at the top left of Figure 1 is available as gear.gif on the homework web page. Write Matlab code that yields images similar to the other three images in Figure 1. This may involve as much trial and error as you wish (in other words, your code needs to work only on this specific image, and need not be general). However, you should follow these rules:

- Use the Matlab function graythresh, which implements Otsu’s method, to compute a threshold \( t \) for transforming the input image \( img \) into a logical image as follows:

  \[
  t = 256 \times \text{graythresh}(img);
  \]
  \[
  \text{bin} = \text{img} > t;
  \]

- Use only built-in morphological operators, connected-components Matlab routines, and logical operators. In particular, you should write no code that explicitly loops over any part of the image. As a convenience, you may use the function selectByArea, provided on the class web page. This function can be called as follows:

  \[
  \text{[cimg area label]} = \text{selectByArea}(\text{img}, \text{areaRange});
  \]

  The input \( \text{img} \) is assumed to be a logical image (that is, its pixels have values either true or false), and \( \text{areaRange} \) is a vector with two integer entries, with \( \text{areaRange}(1) \leq \text{areaRange}(2) \). Either entry can also be set to Inf (which represents \(+\infty\)) or -Inf \((-\infty\)). The vector \( \text{areaRange} \) can also be set to the empty vector [], or be omitted altogether. The latter cases are equivalent to setting \( \text{areaRange} \) to \([-\infty \, \infty\]).

  The output \( \text{cimg} \) is set to a logical image that represents all the connected components in \( \text{img} \) whose areas in pixels are between \( \text{areaRange}(1) \) and \( \text{areaRange}(2) \). The vector \( \text{area} \) has as many entries as there are regions in \( \text{cimg} \), and \( \text{label} \) is an integer image that assigns a different value to the pixels of different regions.

- If you use morphological operators, only use imerode and imdilate. Do not use bwmorph. If you use imerode and/or imdilate, all calls to these functions should use the same structuring element (built once through the Matlab function strel).

- For connected components, you may use either bwallabel or bwconncomp, plus any of the functions used in selectByArea.

- If you use selectByArea, then the smaller entry in \( \text{areaRange} \) cannot be greater than ninety percent of the value of the greater entry, so the following must be true:

  \[
  \text{areaRange}(1) \leq 0.9 \times \text{areaRange}(2)
  \]

  This requirement prevents you from picking a single connected component by entering its exact size. Of course, any finite number is to be considered less than ninety percent of Inf. Other than this, it is OK to select regions by area through trial and error. Typically, you would run selectByArea with no bounds on the areas selected, look at a histogram (help hist) of the resulting areas, and guessing from there which area ranges you are interested in. You would then run selectByArea again, with an appropriate range. When you hand in your code, only include the run with the range specified.

Hand in your code and the three images for support, rivets, and gear, as in Figure 1.

Your results will be evaluated by compliance with the rules above, simplicity of your code, and quality of the results. Results are good when the parts in your outputs are separated into three images as in Figure 1; the regions do not have too many holes, spurious appendages, or stray components; and the shapes are not overly distorted. Some amount of shape distortion is unavoidable. For instance, the results in Figure 1 would earn full credit, even if some of the teeth in the gear wheels are somewhat marred. You may be able to do better, but this may be hard.\(^1\)

\(^1\)On the other hand, I am sure that many of you are champing at the bit for an opportunity to beat your prof!
Figure 1 An image of hand-drawn gear (top left) and approximate regions corresponding to the gear support (top right), three rivets (bottom left) and the main gear wheels (bottom right).
2. In the class notes on Shape Moments (available on the syllabus web page if you misplaced your copy), the following formula is given for the second-order moment matrix of a planar shape:

\[ M = \frac{1}{A} \sum_{p \in R} qq^T \]

where \( q = p - c \) are the pixel coordinates of the points in the shape expressed relative to the shape's centroid \( c \). This formula seems to imply that \( M \) is a scale invariant, because of the following pseudo-argument:

*If you double all dimensions, then the area \( A \) increases by a factor of 4, while the coordinates in \( q \) increase each by a factor of 2. As a result, the fourfold increase in the product \( qq^T \) cancels the fourfold increase in \( A \).*

The conclusion of this pseudo-argument is inconsistent with the statement that \( M \) represents an ellipse that approximates the given shape: If this is the case, then when the shape doubles, \( M \) should change to reflect this growth. Examples given in class show that the size of the ellipse represented by \( M \) does indeed change when the image is scaled up or down.

Where is the flaw in the pseudo-argument above?

3. Smoothing a gray-level image with a box filter, as described in Shapiro and Stockman in Section 5.4, replaces each pixel with the mean value in a rectangular neighborhood around it, so this could also be called a mean filter. The median filter replaces each pixel with the median in the neighborhood instead. Both mean and median are ways to summarize a distribution of values with a single number. A third way would be to compute the mode of the distribution, instead of either mean or median. Recall that the mode of a discrete distribution is the value that occurs most frequently. Let us call the resulting filter a mode filter.

(a) Would you expect the results of applying a mode filter to an image to be more similar to those of a mean filter or to those of a median filter? Why?

(b) Discuss possible advantages or shortcomings of the mode filter, compared to whichever filter you chose in your previous answer. Only consider quality of the results, not computational efficiency. Is a mode filter a good idea?

(c) Implement a mode filter with a 5 x 5 window, and run it on the image noisy.jpg, available on the class web page. In your code, you may assume that the image has values between 0 and 255. It is OK if your output image is smaller than the input image (in other words, do not do anything fancy at the boundaries). Hand in your code\(^2\) and your resulting image. Note: Matlab is clearly not the best language for this type of code, so inefficient implementations are OK. That is why noisy.jpg is so small. If you use the Matlab mode function, you must first convert your image to double as follows:

\[ \text{img} = \text{double}(\text{img}); \]

(d) From your result, show one or two examples of the advantages or shortcomings you discussed earlier for the mode filter. Just say what image (row, column) coordinates to look at, or describe where to look. Then state what that tells you.

\(^2\)Just paste it into your document, do not send me a separate .m file.